Aalto University
School of Science

Juha Selkäinaho

Web Portal for Home Buyer's Selections

Master's Thesis
Espoo, November 24, 2017

Supervisor: Professor Eljas Soisalon-Soininen
Advisor: Mikko Halttunen M.Sc. (Tech.)

Aalto University
School of Science

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Juha Selkäinaho		
Title:	Web Portal for Home Buyer's Selections		
Date:	November 24, 2017	**Pages:**	vi + 49
Major:	Computer Science	**Code:**	SCI3042
Supervisor:	Professor Eljas Soisalon-Soininen		
Advisor:	Mikko Halttunen M.Sc. (Tech.)		

Customer satisfaction is one of the most important factors in home building industry for a successful business. Furthermore, customer service is the most important component affecting home buyer satisfaction. Visiting a design center usually takes 5–7 hours for each customer to select all the interior options for the whole house. Time would be saved and frustration avoided if the options could be reviewed beforehand.

The goal of this project is to implement a customer portal for home buyers to select options for their homes. Before the web portal can be implemented, a literature review is required about the technologies needed for the implementation. Several technologies are researched about web application framework, application programming interface, user interface design and security.

This thesis presents a design and implementation for a web portal using modern technologies. React is selected from the frameworks for the implementation because of its performance and suitability for small projects. In addition, Web API is used as the application programming interface due to its evolvability, flexibility, performance and ease of use.

User interface design tips and guidelines are given about website design, navigation and page design. The page navigation guidelines proved to be the most useful of the tips for this project. The security part of this project reviews common security risks, access control, token-based authentication and single sign-on.

The technologies selected for the implementation proved to be appropriate for this project. Thus, they can be recommended for anyone implementing a web application.

Keywords:	Web API, REST, .NET Framework, React, security, user interface design
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Juha Selkäinaho

Työn nimi:
Web Portal for Home Buyer's Selections

Päiväys:	24. marraskuuta 2017	**Sivumäärä:**	vi + 49
Pääaine:	Ohjelmistotekniikka	**Koodi:**	SCI3042
Valvoja:	Professori Eljas Soisalon-Soininen		
Ohjaaja:	Diplomi-insinööri Mikko Halttunen		

Asiakastyytyväisyys on yksi tärkeimmistä tekijöistä liiketoiminnalle kotien rakentamisessa. Asiakaspalvelu on tärkein kodinostajan tyytyväisyyteen vaikuttava asia. Asiakkaiden vierailuun suunnittelukeskussa ja taloon kuuluvien lisävarusteiden valintaan saattaa kulua aikaa viidestä seitsemään tuntia. Aikaa säästettäisiin ja turhautumiselta vältyttäisiin, jos lisävarusteita voisi tarkastella etukäteen.

Tämän hankkeen tavoitteena on kehittää asiakasportaali kodinostajille talon lisävarusteiden valintaan. Ennen kuin verkkoportaali voidaan toteuttaa, siihen tarvittavista tekniikoista täytyy tehdä kirjallisuustutkimus. Useita tekniikoita tutkittiin liittyen web-sovelluskehykseen, ohjelmointirajapintoihin, käyttöliittymäsuunnitteluun ja tietoturvaan.

Tämä tutkielma esittelee suunnitelman ja toteutuksen verkkoportaalille käyttäen nykyaikaisia tekniikoita. Toteutukseen valitaan sovelluskehyksistä React johtuen sen suorituskyvystä ja sopivuudesta pieniin projekteihin. Ohjelmointirajapinnoista toteutukseen valitaan Web API, koska se on helposti jatkokehitettävä, joustava, suorituskykyinen ja helppokäyttöinen.

Käyttöliittymäsuunnittelusta annetaan ohjeita ja vinkkejä verkkosivuston suunnitteluun, navigointiin ja verkkosivun suunnitteluun. Verkkosivun navigointiin liittyvät ohjeet osoittautuivat hyödyllisimmiksi käsitellyistä vinkeistä. Tietoturva osio käsittelee yleisiä tietoturvauhkia, pääsyn hallintaa, tietuepohjaista todennusta ja kertakirjautumista.

Toteutukseen valitut tekniikat havaittiin sopivaksi tähän projektiin, ja niitä voi suositella myös muihin verkkosovelluksiin.

Asiasanat:	Web API, REST, .NET ohjelmistokehys, React, tietoturva, käyttöliittymäsuunnittelu
Kieli:	Englanti

Acknowledgements

This work was done for Kova Finland Oy as a project that was requested by home builders. I wish to thank Jouko Väkiparta, the CEO of Kova Finland Oy, for the opportunity to write this thesis. In addition, I want to thank my supervisor Professor Eljas Soisalon-Soininen for his guidance and feedback in writing this thesis. Furthermore, I wish to thank my instructor M.Sc. (Tech) Mikko Halttunen for his help during the implementation and comments about the thesis. Moreover, I would like to thank my family and friends for their support during my studies and this thesis.

Espoo, November 24, 2017

Juha Selkäinaho

Contents

Chapter 1

Introduction

Home building is a customer-oriented business – home builders market and sell homes directly to the customers. Customer satisfaction is one of the most important factors in home building industry for a successful business [43]. Moreover, customer service is the most important component affecting home buyer satisfaction. However, there is room for improvement in customer service of home builders.

A lot of marketing of new homes is already done in the web. 96 % of home buyers aged 18–44 used the Internet to search for a home in 2011 [11]. In addition, customers are more inclined to research a product online instead of visiting a sales office and spend more time viewing the information about each product online. Therefore, home builders should adopt more online services to improve customer satisfaction.

In the United States home building market, the process of selling a home to a customer starts when the customer walks in to a sales office and chooses a model for a home and a lot in a community on which to build the home. At this point, the customer can select exterior options for the home. In addition, a contract for the new home is written but it does not need to be signed yet. The customer can still return to change the exterior options. Before moving forward, the contract needs to be signed. Afterwards, the customer goes to a design center to select interior options, e.g., kitchen cabinets, flooring materials, bathroom shower fixtures and wall tiles.

There are plenty of different options to choose, for example, kitchen cabinets might have five different wood types, each wood could have ten different carving styles and 20 different finish options. So, selecting all the interior options for the whole house can be time consuming and visiting a design center usually takes 5–7 hours for each customer [35]. During that time, the customers can get frustrated selecting all those options and that might leave a bad image about the company. In addition, a study [45] found that

this decision fatigue can lead to reduced self-regulation and active initiative. This means reduced physical stamina, less pain tolerance, reduced persistence in the face of failure, difficulty with arithmetic calculations and greater passivity.

To battle this problem, the home builders would like to have another step between sales office and design center to market the design center options. If the customers could browse the design center options beforehand in the web, they could have a better vision of what they want to buy. This would save time in the design center for the customer and the home builder. In addition, the customers would be happier which is good for the business.

The focus of this project is to develop a web portal for customers to view and choose interior design options for a home before going to the design center. Those options would be saved as favorites to the database and they would be shown in the option configurator application when the customer goes to design center to look at the materials and write a contract for those options. The Portal needs to be implemented in a way that makes it extendable to replace the current option configurator application.

To develop this web portal, we will first need to review available technologies. These technologies are related to the client side and server side and transferring data between them. Web API is a useful technology for providing data from the server to the web user interface. It is also important to study the user interface design in application development to develop a modern and user-friendly user interface. There are certain security aspects required in developing a web portal from the stand point of a home buyer or a home builder. For example, home builders require security that restricts access to their business sensitive data, and home buyers want to keep their personal data secure.

In chapter 2, we will start by reviewing some of the technologies needed in web application development. Chapter 3 introduces technologies related to web application programming interfaces. In chapter 4, we talk about user interface design. Chapter 5 explains some of the security aspects. After this literature review, we will describe the current system and the new requirements in chapter 6. Chapter 7 will show the design and the implementation of the web portal. In chapter 8, we will evaluate how well the implementation satisfies the requirements. Finally, in chapter 9 we will discuss about what was done in this project.

Chapter 2

Web Application Development

In this chapter, we review some of the technologies required in web application development. First, we shortly review HTML and JavaScript languages used in all modern websites. Second, we talk about web application framework called ASP.NET. After that, we present the Model-View-Controller design pattern. Finally, we dig into two client-side frameworks used in web application development, Angular and React.

2.1 HTML

HTML is a markup language which is used to define the structure of the web page [24]. HTML consists of different tags that correspond different elements of the page, such as headings, paragraphs, tables and forms. These elements can contain other elements, text and attributes which are name–value pairs that give information to an element.

HTML does not contain any programming logic that would change the content. Other methods are required to make the web pages dynamic. Styles or the visual layout of web pages can be done in HTML but it is recommended to do it separately in Cascading Style Sheets (CSS). This keeps the definition of the structure of the page separate from the visual layout.

2.2 JavaScript

JavaScript is an interpreted programming language used in most modern websites [20]. A JavaScript interpreter is included in all modern web browsers making it a client-side programming language. This makes the UI more responsive and frees up the load on the server as some logic can be done at the client side instead of the server.

Document Object Model (DOM) is used to represent the HTML elements as objects in JavaScript [25]. DOM is a tree structure of the element nodes. It allows changing the contents, structure and style of the web page in JavaScript. JavaScript can attach events to certain interactions with the web page making it dynamic. This way, a user can interact with the website.

JavaScript is a powerful tool in web application development but it can be complex to develop everything with just JavaScript. There are many useful JavaScript libraries to ease the development of web applications, such as JQuery [3], Angular [1] and React [4]. JQuery is commonly used to simplify JavaScript development with helper functions, for example, animation functions and function to post data to server. In addition, it greatly helps selecting and changing DOM elements in the web page. Angular and React are described in sections 2.5 and 2.6.

2.3 ASP.NET

As described in section 2.1, simple HTML pages serve static content. Web applications are complex and require dynamic web pages. ASP.NET framework [37] offers this with Web Forms programming model that has Aspx pages which can contain HTML and programming code, such as Visual Basic or C#. When the server gets a request to an Aspx file from a client, ASP.NET runtime processes the file to HTML and JavaScript and sends it to the client in a form that the browser can interpret. The server side code makes the page dynamic by changing the output depending on the input, for example, it can show and hide different elements on the page or access a data source and print that to the page. ASP.NET framework also contains server controls, such as buttons, text fields and grids. They can be accessed in the server side code and are output as HTML elements.

2.4 Model-View-Controller Design Pattern

The Model-View-Controller (MVC) design pattern [41] divides an application into three separate parts. These parts are called model, view and controller. Model part handles data management, view handles presentation and controller handles business logic. User interacts with the view which is the graphical user interface (GUI). Controller processes the user input and sends it to the model component. The view also manages changes in the model component for the user to see.

2.5 Angular

AngularJS [23, 41] is a JavaScript framework which uses the MVC design pattern. As it is a JavaScript framework, it is run client-side in a web browser and does not need to be compiled. It uses a templating system where custom HTML tags and attributes are written inside a HTML document. These custom tags and attributes are directives for the Angular library and are interpreted to regular HTML.

AngularJS uses a mechanism called two-way data binding to automatically save the data to the model when the user updates something in the view. This mechanism also updates the view when data changes in the model. To decide when to update the view and model, Angular uses a method called dirty checking. It is an algorithm which finds discrepancies in variable values between the view and the model. The framework synchronizes the data often and that might lead to performance issues if there is too much data in the view. To avoid this, AngularJS uses a scope variable to define what data to update between the view and the model. Two-way data binding is done only for the variables and function defined in scope. Only the data required in the view should be in the scope since all excess data increases the time to perform dirty checking.

2.6 React

React is a JavaScript framework for rendering web pages through the use of components which define HTML elements [38]. These components can be React JavaScript objects or JSX objects which allows you to use HTML syntax in JavaScript. However, JSX is not valid JavaScript syntax and browsers cannot run it. It needs to be transformed to browser supported JavaScript by a tool, such as Babel [2]. JavaScript can also be used inside the JSX markup to make use of variables, conditions and loops.

Custom components can be created with business logic in them to alter the HTML they render depending on their properties and state. Properties are given when creating the component and the state can be updated. Updating the state will render the changes automatically to the UI. React keeps track of what has changed in the data so the page is not updated needlessly. Only the elements, that are updated in the components, are updated to the page.

These custom components can be used in JSX as any other HTML element. Instead of using just one large component for the whole application, they should rather be split into smaller components which can be reused

throughout the application.

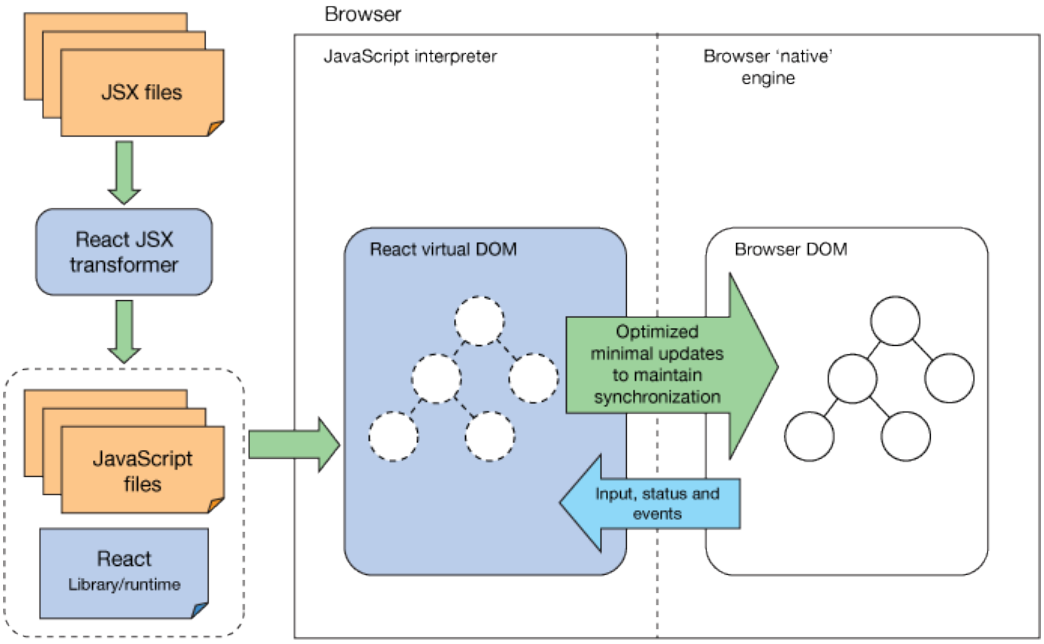


Figure 2.1: React architecture overview from [27].

Figure 2.1 shows an overview of React operations. First, the components are created in JSX files that are transformed into JavaScript files. After that, a user opens a web page in browser which runs the JavaScript files. Then, the JavaScript code updates the React virtual DOM, and React library handles the required updates to the browser DOM. Finally, the browser renders the DOM for the user to see.

Chapter 3

Application Programming Interface

This chapter describes a few application programming interface (API) technologies used in the web. First, we present a technology called Simple Object Access Protocol (SOAP). After that, we describe a technology called web services. Then, we talk about Representational State Transfer (REST) methodology. Finally, we describe Web API calls.

3.1 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a protocol used to define what type of information is transferred between applications [36]. If defines what parameters are used, what are their types and how are they defined in the data that is transferred. It also tells us what is expected to be returned and in what format that is. This information is passed in SOAP messages that are XML. The applications can be written in any programming language that can read and write XML documents.

The idea is that applications can exchange data via messages structured in a way defined by SOAP. The message consists of a header and a body. Header can contain identification and authentication information. Body contains the actual message and it can be any kind of XML. In addition to basic data types, such as strings and integers, also structs and arrays can be defined in the body.

3.2 Web Services

Web Services are application functionality that can be accessed over the network [36]. They provide an interface and an abstraction layer between application code and application client. The only requirement is that a web service must be able to send and receive messages over an Internet protocol. Websites can also be considered as web services since they send HTML data over HTTP from a server to a client web browser.

A Web Service architecture can be described by a stack of layers called discovery, description, packaging, transport and network. The discovery layer is used to get the descriptions of the web services. Description layer contains information about what packaging, transport and network protocols are used in the web service. Web Service Description Language (WSDL) is a standard way to provide the descriptions. Packaging layer defines a format for the transferred data. SOAP is a commonly used format for this. Transport and network layers are used to transfer the data.

WSDL is used to describe what a web service does and how a client can use it. The description can be generated automatically. This enables the implementation of self-describing web services which makes it easier to maintain and use the service. Although, there is no support of different versions in WSDL. So, significant changes in the web service can cause issues if they are not notified otherwise and the client side implementation is not changed.

3.3 Representational State Transfer (REST)

Representational State Transfer (REST) [18, 19] is a web architecture style that aims to minimize the network connection time and, at the same time, keep the system as scalable as possible. REST does this by a set of constraints it sets for the system. These constraints are client–server architecture, statelessness, cache, uniform interface, layered system and code-on-demand.

Client–server architectural style allows the improvement of user interface portability and server scalability by separating the user interface and server concerns. Stateless constraint means that each request must contain all the information required to understand it and it cannot use any stored context on the server. This improves visibility, reliability and scalability. But, it might decrease network performance by sending the same data multiple times over multiple requests.

Cache constraint improves efficiency, scalability and performance by partially or completely removing some interactions. However, the cache can

decrease reliability since the data sent might not be up to date.

Uniform interface improves the visibility and simplifies the overall architecture, although it degrades efficiency since data is transferred in standardized form instead of what is optimal for the application. Uniform interface is achieved with four interface constraints: identification of resources; resource manipulation through representations; self-descriptive messages; and hypermedia is used to describe the application state.

Layered system constraints each component of the application to a single layer. This reduces system complexity. Layers can be used to separate legacy services from new services or improve system scalability with load balancing. Layered system adds overhead and latency data processing and reduces performance.

Code-on-demand improves system extensibility by allowing downloading and executing client side code after deployment. However, this reduces visibility and is regarded as an optional constraint in REST.

3.4 Web API

REST is a commonly used style for designing Web API that is an interface for accessing data using standard HTTP methods and headers [8]. Web API services are easier to use from web browsers and other HTTP clients than SOAP web services. With SOAP web services the client would need to access the WSDL file and interpret it to construct the SOAP request to send.

There are several guidelines for designing Web APIs. They should be browser friendly and accessible from other HTTP clients as well. Web APIs should use different HTTP methods for requests, at minimum, GET method is used for retrieving data and POST method is used for updating data. Other commonly used methods are PUT, DELETE, HEAD and PATCH. Web APIs should use message formats that are readable by client side code or browser, for example, XHTML and JSON. In addition, browser-friendly authentication should be supported.

The Richardson Maturity Model (RMM) is a classification system for Web APIs measuring how well they use web technologies. It is divided in four different levels: Remote Procedure Call (RPC) oriented, Resource oriented, HTTP verbs and Hypermedia. Benefits and trade-offs for each level can be used as guidelines for Web API design.

The RPC oriented level uses HTTP just as a transport protocol and sends all the information in the body of the requests and responses using only a single HTTP method to a single endpoint. SOAP web services are an example of this RMM Level 0.

RMM Level 1, that is the Resources level, accesses multiple resources using a single HTTP method. The operation is represented in the URI. This style offers more evolvability since more functionality can be added to new resources without changing the old resources.

In addition to having multiple resources, RMM Level 2 includes multiple HTTP methods for a single resource endpoint, that is, the operation is conveyed in the HTTP verb. A client could send a GET request to a resource to retrieve data and send a POST request to the same resource to update the data. This allows support for additional HTTP capabilities, such as caching and content negotiation.

RMM Level 3 adds use of hypermedia to the Web API. This means adding links or forms to the response that the server returns. A link in a Web API response is a URI to a related resource and the relationship between the current resource and the related resource. The client only needs to know a root URL to access the API and navigate the rest by using links from the responses. The resource URLs could change or be added on the server and existing implementation of clients would still work. The trade-off is that implementing a hypermedia system is more difficult than the other levels.

Examples of Finnish APIs are reviewed in a study [21]. City of Helsinki offers a variety of public Web APIs [13] to convey data stored in the information systems of the city, for example, there are Web APIs about statistics, meeting decisions and location of snowplows. A company called PlanMill offers a Web API for their clients to access their project management, ERP and CRM products [33]. All of these examples of Web APIs are developed using the REST methodology.

Chapter 4

User Interface Design

This chapter reviews some methods for designing user interfaces specifically for websites. First, we review general website design methods. Second, we talk about designing navigation for websites. Last, we describe visual page design methods.

4.1 Website Design

There are four rules in website design: keep it simple, keep it consistent, keep it current and keep navigation to three clicks [10]. Keeping it simple means that the layout should be as easy as possible. In addition, in western countries the eyes of the reader follow from left to right and from top to bottom. Interface should be consistent on all pages of the site or the user may feel confused that they are on a different site on different pages. You should keep the information on the site up to date. If the user can see an old date or feels that the information is not current, they have no reason to return to the site for new information and may feel that the company is not serving its customers properly. Users should be able to find what they are looking for in three clicks.

Cooper et al. [15] begin designing the user interface of a digital product by pondering what posture is appropriate. Posture of a product is how it presents itself to users. Web applications can have sovereign or transient posture. Sovereign web applications should be full screen applications with a lot of controls and data objects. Related functions and objects should be grouped together in panes or other regions. Instead of navigating from one page to another, the view should be a single page with minimal re-rendering of information. Transient web applications aim to provide better access to occasionally used information and functionality. As the user interface is

used rarely or only once, it is important to provide clear orientation and navigation.

4.2 Navigation

Cooper et al. [15] also offer tips about designing navigation for websites. There can be multiple levels of site navigation and content navigation. Primary navigation means how the user can get to the major sections of a website. These navigation links are usually on the top or left side. Top navigation is better than side navigation because side navigation makes the page crowded and makes it harder for the user to see past it to read the content. On the other hand, there can be more and longer items on the left side navigation and they are easier to read. Navigation controls can also be hidden and shown dynamically to save space on the page.

If top side navigation is used on the primary level, second and third level navigation can be done with left side menu or another row of horizontal menu. Another approach is to use fat navigation. This means that clicking the primary navigation reveals another large modal menu with navigation choices. It can be larger than a regular menu because it is shown only temporarily. With multiple levels of navigation, it is important to provide feedback on where the user is in the navigation structure. This can be done with a visual change in the navigation element or breadcrumbs which are links to each level of the navigation path.

Content navigation means navigating through series of contents, such as photos and articles. Usually these are shown as different kinds of listings, for example, article headlines or image galleries. Modern way to present these is the feed format used in blogs and social media. The content lists are commonly organized in multiple different ways, such as by date or by topic. It is also common to show multiple navigation schemes to browse the content.

4.3 Page Design

According to Lynch and Horton [28], the primary purposes of graphic design are to create a visual hierarchy, define functional regions of the page and group related page elements. Visual hierarchy is created with contrast to see with a glance what is important on the page. Related page elements are grouped together for the user to see structure in the content.

There are some visual design principles to help web page design [28].

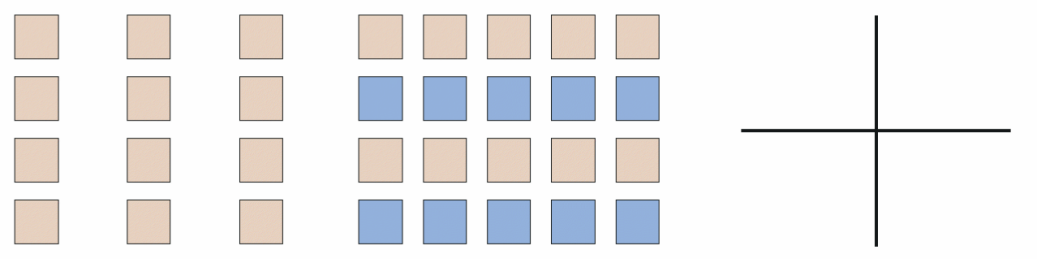


Figure 4.1: An image showing proximity, similarity and continuity design principles from [28]. First part of the image shows how proximity of elements makes the image appear as three separate columns. Middle part of the image represents related rows because of similar colors. Last part of the image shows how it seems more like two lines crossing instead of four lines meeting due to continuity.

These are proximity, similarity, continuity, closure, figure-ground relationships and uniform connectedness. Proximity means that elements that are closer together seem related to each other more than elements that are further apart. Visually similar elements are considered as a group. People tend to see continuous contours and paths rather than separate. Examples of proximity, similarity and continuity are depicted in figure 4.1. Closure means that people are biased to see completed figures even when the contours of the figure are broken. In figure-ground relationship, the user can see the image in a different way depending on the relative size of the figures. Relatively small elements will be seen as the figure and the larger field as the ground around the figure. Figure 4.2 shows examples of closure and figure-ground relationship. Uniform connectedness means the relations of elements enclosed within other elements.

Another important aspect in page design is to get user's attention effectively without annoying the user. A study [42] found that while noticeability usually implies annoyance, there are a couple of attention grabbers that are more likable than others while still being able to get the user's attention. A single pulse of a shadow around a message box is as noticeable as a pop-up but much more likable. Pop-ups provide good noticeability but they should be integrated with the page to minimize the annoyance. The most likable attention grabber was an icon with badge but it was not very noticeable. An example of the icon with a badge attention grabber would be a letter icon with number appearing on the corner of it.

Figure 4.2: An image showing closure and figure-ground relationship from [28]. First image shows how we are biased to see a white square instead of four broken circles. Second and third image are showing an example of the figure-ground relationship. The second image seems more like two faces and the third image seems more like a vase as the relative sizes change.

Chapter 5

Security

A web portal requires means to identify the user and to check that the user can access this information. It is also important to keep the portal and information secure from harmful attacks. First, we list some common security risks. Then, we explain what access control means. After that, we describe the token-based authentication. Last, we talk about the single sign-on technology.

5.1 Common Security Risks

The Open Web Application Security Project lists the ten most critical web application security risks [44]. These are injection, broken authentication, sensitive data exposure, XML external entity, broken access control, security misconfiguration, cross-site scripting, insecure deserialization, using components with known vulnerabilities and insufficient logging and monitoring. We will focus on these in the next subsections.

5.1.1 Injection

Injection attack means injecting harmful code to be run as a part of a procedure the system runs [44]. Injection flaws can be found on different queries, such as SQL, Lightweight Directory Access Protocol or Object-relational mapping. SQL injection is the most well-known and the only one that affects this project.

In SQL injection attack the user input can be appended or included in the SQL query that the system runs [14]. SQL injection can be a risk if the user input is not sanitized and it is used directly in SQL query. It can be used to return restricted information from the database. It can be also used

to bypass login to an account.

SQL injection can be prevented by using parameterized queries or escaping the special characters in the user input [14]. If the query is parameterized, the SQL command engine that runs the query uses a single field from user input only as a certain type parameter and it cannot be interpreted as a part of a SQL query. If parameterized queries cannot be used, special SQL characters need to be escaped so that they cannot cause a SQL injection attack. It is safer to use a library designed to escape the special characters in the SQL framework that is used. Otherwise, it is difficult modify the user input to be safe for the SQL command engine.

5.1.2 Broken Authentication

Broken authentication means that an attacker may gain access to the system by using an account of another user [44]. This can happen if the system allows automated attacks, such as brute-force attack or using a list of valid user names and passwords breached from another system. Using multi-factor authentication can prevent such automated attacks. Furthermore, the system should also check for weak passwords. In addition, a modern hashing function should be used to store the passwords instead of storing them in plain text, encrypted or weakly hashed. Moreover, forgot password process should not use knowledge based answers since those can not be guaranteed to be safe.

Authentication system with too strong security may hinder usability [22]. Passwords should not be weak but remembering long and complex passwords is difficult for a user. The user must remember several passwords for various systems. However, the user might be tempted to use the same password for multiple sites or a weak password that is easier to remember but also easier to crack. One suggested solution is using single sign-on. This is discussed in section 5.4.

5.1.3 Sensitive Data Exposure

Data needs to be secured in transit and when stored to avoid exposure of sensitive data, for example, personal information, credit card numbers and passwords [44]. The data must be classified to find out if it falls under a privacy law, such as the EU General Data Protection Regulation. The sensitive data that is stored needs to be encrypted, and all transmitted data needs be secured by using HTTPS. Furthermore, the data should also be discarded from storage once it is not needed anymore. In addition, cache should be disabled for responses that contain sensitive data. Proper authorization can also prevent sensitive data exposure [39].

Target stores suffered this attack in 2013 [39]. Information of 40 million credit cards were stolen and personal information of 110 million people might have been compromised. This personal information includes email addresses and phone numbers. The credit card data was not encrypted in the memory of the point-of-sale devices and was stolen from there.

5.1.4 XML External Entities

In XML External Entities (XXE) attack an entity tag is inserted in a XML file that is sent in a request to an application which accepts XML documents [44]. The entity tag can output a file from the server to a variable that can be used in the XML. This way the attacker can retrieve sensitive information if that variable is also shown in the response. The XXE attack can also be used as a denial-of-service attack if the file loaded by the entity tag is large. To prevent this attack, processing XML External Entities and Document Type Definitions should be disabled in the XML parsers used by the system. In addition, the format of the incoming XML should be validated, for example, by XML Schema Definition (XSD). Furthermore, if the system uses SOAP as discussed in section 3.1, it should be upgraded to the latest version or at least version 1.2. Also, the XML processors and libraries should be upgraded.

5.1.5 Broken Access Control

Broken access control means that users are able to access data that is not intended for them [44]. Vulnerability can lead to disclosure of unauthorized information or modification or removing of data. This can happen if the access control checks can be bypassed by modifying the URL, application state, the HTML page, access control token, cookie or hidden field. To fix this, access control should be enforced by server side code so that client side modifications do not affect it. Furthermore, access should be denied by default except for public resources, such as login page. In addition, access control should be enforced for Web API methods PUT, POST and DELETE. Moreover, the rate of accessing the API should be limited to minimize the risk of an automated API attack tool. Also, access control failures should be logged and repeated failures reported.

5.1.6 Security Misconfiguration

Security Misconfiguration means that the web application is at risk because some part or library of the application has been configured so that it is vulnerable to an attack [44]. To avoid this risk, all unnecessary features

should be disabled or removed, e.g., ports, services and accounts. In addition, default accounts and passwords should be changed. Furthermore, error messages should not reveal information about how the system works. Moreover, configuration of a library should be updated when the library is so that old vulnerable settings are not used. Also, the security settings of server applications, frameworks and databases should be reviewed that they are secure.

Security misconfiguration can lead to big breaches, for example, four million personnel records were stolen from a federal agency called the Office of Personnel Management in 2015 [39]. A report half a year earlier stated that they did not review the security of their systems regularly.

5.1.7 Cross-Site Scripting

In cross-site scripting (XSS) attack unsafe JavaScript is executed in the user's browser [44]. The JavaScript code can come from the user's input. If the application runs unsanitized user input, the attack is called reflected XSS. If the unsanitized input is stored and run later by a different user, the attack is called stored XSS. Common XSS attacks include session hijacking, account takeover, key logging and malicious software downloads through the victim's browser. XSS attack can be prevented by using safe frameworks that escape user input automatically, e.g., Ruby 3.0 or React JS. In addition, request data should be escaped based on the context in the HTML output.

5.1.8 Insecure Deserialization

Applications that need to store their state on the client or the filesystem might be vulnerable to insecure serialization [44]. The attack is possible when the state is stored in binary format, such as Java Serialization, or text formats, e.g., Json.Net. A few conditions must pass for the application to be vulnerable. First, the system can serialize arbitrary data types, Second, there are classes that can alter the application behavior when deserialized. Third, the system deserializes objects sent by an attacker.

The only fully safe way to prevent insecure serialization is to only allow serialization of primitive data types [44]. Otherwise, the serialized objects should be encrypted or security checked. In addition, the serialization code should be isolated and run in a low privilege mode. Furthermore, deserialization errors and frequent attempts should be logged.

5.1.9 Using Components with Known Vulnerabilities

Using components with known vulnerabilities is an issue because the vulnerabilities are known among attackers and there are ready-made attacks for them [44]. You should know the versions of all the components in the system. Furthermore, you should research for vulnerabilities in the software. If a vulnerable component is used, it should be updated to a fixed version as soon as possible. The research and update process should be continuous. If the patching is a monthly process, the components can be vulnerable for days unnecessarily.

5.1.10 Insufficient Logging and Monitoring

Insufficient logging and monitoring is a risk because attacks are not detected fast enough so that the attackers have time to achieve their goals [44]. All logins, access control failures, input validation errors and high value transactions should be logged. Furthermore, the logs should be monitored for suspicious activities. In addition, larger organizations can benefit from real time alerts and automatic responses to attacks, such as blocking attack requests.

5.2 Access Control

Access Control means the way of restricting access to information to only those who are allowed to access it [26]. It is divided into two parts: authentication and authorization. Authentication is a way to identify the user. Authorization checks that the user is allowed to access the information.

Authentication can be based on three different methods: what you know, what you are or what you have. Combination of username and password is an example of what you know. Biometrics are an example of what you are, for example, finger prints or face recognition. These are recognized by a probabilistic algorithm which can lead to errors. An example of what you have is a security token which we will discuss in the next section 5.3.

There are a few security policy models for authorization, for example, Mandatory Access Control (MAC), Discretionary Access Control (DAC), Role-Based Access Control (RBAC) and Originator-Controlled Access Control (ORCON) [26]. In MAC, the system determines which users can access what. In DAC, the user decides who can access the information the user controls. In RBAC, access is given based on the role the user has, for example, a user with a system administrator role can access all files but a user

with a customer role can only access his own files. In ORCON, the creator of the document determines who can access the document and under what conditions. It is very difficult to enforce in a distributed system, for example, Digital Right Management would not be an issue if the enforcement was reliable.

A study [12] notes some common security issues with private web APIs. The application should limit the frequency of invalid authentication attempts from the same IP address or for the same username to avoid brute force attacks. Web APIs should only use HTTPS to access the service to secure the transmitted data, such as passwords or security tokens.

5.3 Token-Based Authentication

Token-based authentication means authenticating to a web service by sending a security token along the request to the server [8]. The security token can contain the security information to identify the user or it can just be a reference to that information. If the token is a reference, they can be shorter and easier to revoke. Shorter token is beneficial when it is embedded in the URI. Revoking a reference token just means deleting it from the token storage. However, obtaining the security information stored in the token is more difficult if the issuer of reference token is not the same as the consumer. The reference token can be a random bit string that is long enough to be unique. The stored security information should be identified by the hash value of the reference token. Computing the store key from the token is easy but computing a token from a store key is hard. This provides extra security against an attacker who gets access to the store keys.

If the token contains the security information, the token consumer does not need to access an external storage. The downside is that they require cryptographic methods to encrypt the information and that might make them longer. The Security Assertion Markup Language (SAML) is a commonly used method for self-contained tokens for web services. SAML represents the security information in XML and it is protected with XML-Signature and XML-Encryption.

5.4 Single Sign-On

Single Sign-On (SSO) service allows the user to authenticate to multiple different services with one identity verification [40]. This way the user only needs to remember one username and password to access multiple services.

SAML has been used for SSO technologies, for example in Shibboleth [5], which is used by all universities in Finland. Facebook uses SSO that is based on OAuth technology. Google provides SSO service that is based on OpenID Connect.

OAuth was originally developed to provide access rights to resources for third-party applications [40]. For example, if a user has an account in a photo sharing service and wants to share some photos to a printing service through their website, OAuth can be used to give access to certain photos without giving full access to the user's account. OAuth has also been used for user authentication to third-party services. Facebook uses an identification token in addition to OAuth 2.0 protocol to provide authentication [9].

OpenID Connect was standardized in 2014 [16]. It is based on OAuth 2.0 protocol instead of the OpenID protocol. It was created to provide authentication to the OAuth 2.0 protocol with additional features, such as dynamic client registration and discovery extension. The OpenID Connect protocol is used authenticate users to a relying party (RP) using their existing account at an identity provider (IdP) or more specifically OpenID provider, for example, users can log into TripAdvisor by using their Google account.

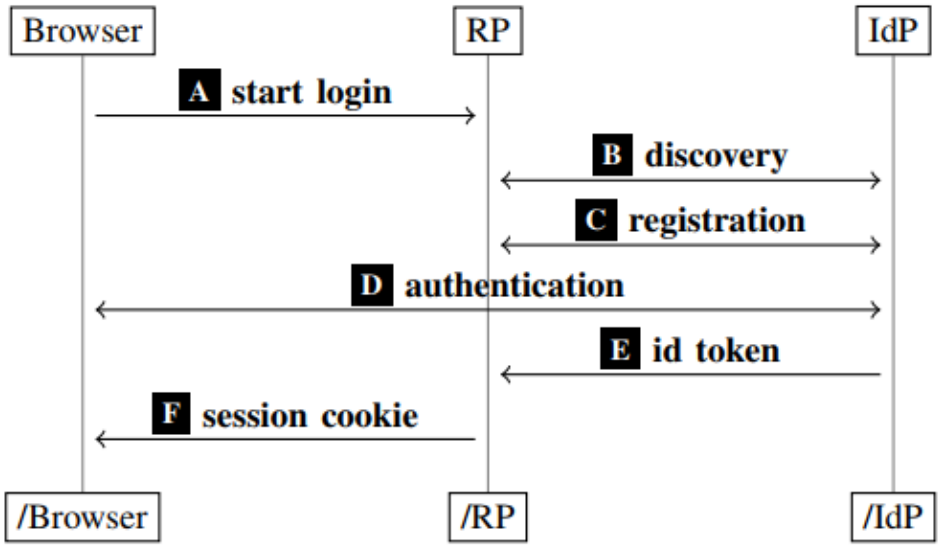


Figure 5.1: OpenID Connect protocol flow from [16].

Figure 5.1 shows an overview of the protocol flow of OpenID Connect. First, the user start logging in at a relying party, for example, TripAdvisor.

Then, the relying part uses the discovery extension to fetch operational information and dynamic client registration extension to register to the IdP, for example, Google account. After that, the user authenticates to the IdP, for example, by entering password for the user's account. Next, the IdP sends the identification token to the RP to verify the identity of the user. Finally, the RP sets a session cookie to the user's browser to complete the login process.

Chapter 6

Current System and New Requirements

This chapter describes the state of the current system and what are the requirements for the new implementation. In the architecture section first, we describe the existing web services. Second, we talk about the current Web API implementation. Third, we describe the existing configurators and how they are used to select options. The last part of the architecture section lists the data model relevant to this project. In the following section, we first give an overview of the project. Then, we describe different kinds of requirements for this project.

6.1 Current Architecture

Our company has been developing software solutions for home builders in the USA since 2003. The development of the main solution Builder Portal has been ongoing since the beginning of the company and has had several architectural rewrites to keep it up to date. Oldest parts of the system still date to the version that was developed using the first beta versions of Microsoft .NET Framework. The main modules of the system are the main Enterprise Resource Planning (ERP) portal, a web application and its web services, marketing tool web application Web Pro for configuring a house on a website, sales configurator windows application for the back-office users and mobile applications.

6.1.1 Web Services

The ERP software uses Web Services as implemented with the framework of ASP.NET. The web services are currently used by other applications to communicate with the ERP system. Some of the applications are helper applications for the ERP system developed by the same company, for example, mobile applications, file manager and sales configurator. Some are used by sister companies with applications closely integrated to the ERP system, e.g., a CAD application that uses options from the ERP system. None of the web services are publicly open to home builders. The ERP system also uses web services of third party applications to communicate with them, for example, a document signing service that sends documents in the ERP portal to vendors to get electronic signatures.

6.1.2 Web API

The Web API is also implemented with the ASP.NET framework. It is only used by the home builders on their websites. For example, there are calls for inserting new sales leads to the system and getting the communities and models used in the system. The CAD software, that has used web services to connect, has more recently started using the Web API to get the option data. The Web API returns response of objects in JSON format. If data needs to be sent to the calls, the request body is also in JSON format.

6.1.3 Existing Configurators

There are currently two configurators used by the system, the web configurator and the sales configurator. The web configurator is a portal used by new customers on the home builders website. There the customers can select a model, view some of the sales office options, print a brochure and leave their contact information. There are two versions of the web configurator. The older one uses web service calls to interact with the ERP system, and the newer one uses the Web API calls.

The sales configurator is a smart client application that is downloaded from the ERP system to a client machine and run on it. It is used by sales people in their office to select sales office and design center options with the customer. It is also used to generate and print contract based on those selected options. The sales configurator communicates with the ERP system using web services. It was designed at a time when internet connections were not fast or reliable enough. Instead of waiting after each interaction, it was better to wait a longer period at the beginning. So, the sales configurator

downloads whole data set of options in the start and allows the user click through them without any interaction to the server. After the user is done selecting the options, the selected data is sent back to the server.

6.1.4 Data Model

The ERP system has data set up in a hierarchical structure of Business Units, for example, areas, divisions and cities. Leaves of the Business Unit tree are called Communities, which are areas of a neighborhood where a home builder owns the land and builds homes. Each community has a set of Models available. Model is a plan for a home that has a list of options to choose from.

These options are called Option Selections. Some examples of Option Selections would be kitchen cabinets, foyer flooring materials, fireplace, sunroom or ceiling fans. They are divided into two different sets, sales office and design center options. Option Selection are further grouped into different Locations and Categories. Examples of Location might be kitchen, master bedroom, garage and dining room. Examples of Categories could be cabinets, plumbing, flooring, fireplaces and appliances.

Option Selections are also divided into four different types, binary, quantity, attribute and radio. Binary options have true or false values for an option, for example, whether a sunroom is selected or not. Quantity options define an amount of option to select, e.g., number of ceiling fans. Attribute and Radio options have a predefined list of values to choose from. These values are called Option Values that can have a further defining style and color. For example, a fireplace could be a gas or a wood fireplace and a customer could choose different style and color for a mantel in the fireplace. Option Menus define which Option Values are shown for an Option Selection.

Examples of Cabinet Category Option Selections in the sales configurator are showed in image 6.1. Main Kitchen Base Cabinet Selection, Main Kitchen Wall Cabinet Height and 36” Cabinet Over Refrigerator are all Option Selection that are grouped in Location Kitchen. Cabinet Hardware is a binary Option Selection that is shown as a checkbox. Quantity Option Selection would be similar but instead of a checkbox there would be a text field with a number. Main Kitchen Base Cabinet Selection and 36” Cabinet Over Refrigerator are examples of attribute Option Selections. For the first one, Cabinet Level 5 is an example of an Option Value that has sets of style and color from which to choose. Attribute Option Selections are shown as drop-down lists. 36” Cabinet Over Refrigerator has a drop-down that is open. Main Kitchen Wall Cabinet Height is an example of a radio Option Selection. It is shown as radio buttons where only one of the values can be

selected.

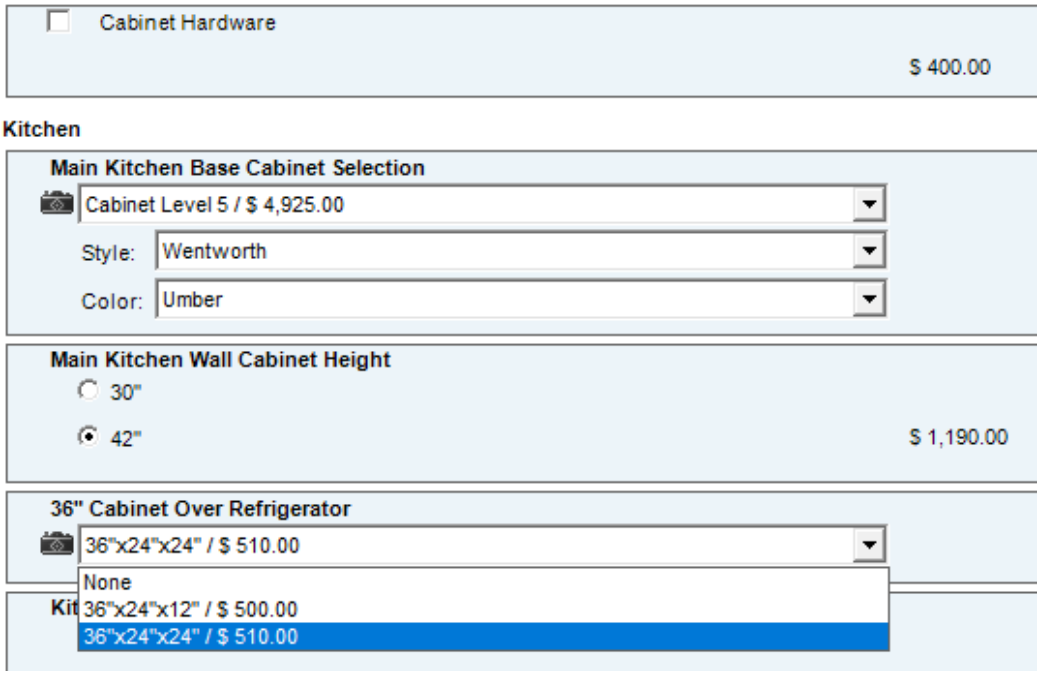


Figure 6.1: Image showing different types of options in the sales configurator.

Customer object is a record of the contact information for a customer in the system with a status of prospect, under contract or home owner. Lead is a potential customer who might have interest in buying a home in a community. Some home builders do not use Leads but only Customers. One could define difference of Leads and Customers such that Lead becomes a Customer once they walk into a sales office.

When a customer signs a contract to buy a home, the system creates a Sales Order. It contains information about the Model the customer has selected and the Community in which the home is built. It also has a list options the customer has selected for the home.

6.2 Requirements

The way that this project fits into the process of a customer buying a home in the ERP system is depicted in figure 6.2. The first step is customer selecting a Model in a Community and sales office options. These are saved in a Sales Office Worksheet object in the system. Sometime after that in step

two, the customer signs the contract, and the system creates a Sales Order. The third step happens about a month later when the customer goes to the design center and selects the design center options. In the design center the customer can see the actual materials, such as cabinet doors, countertops and backsplash tiles inside a kitchen. The web portal created for this project falls in between steps two and three.

The customer should get a link, username and password to login to a customer web portal. That portal contains information about the home of the customer. It also has a link to a design center portal that is the portal which is created for this project. The login page could also redirect the customer directly to the design center portal if the home builder does not want to create a customer portal to view home information.

The customer interacts with the design center portal and selects her favorite options. The favorite choices are saved to the database of the ERP system. Now when the customer visits the design center studio, a list of the favorite choices can be printed, and the customer can see her favorite materials first without going through all the available choices. The design center representative can also see the favorite choices in the sales configurator.

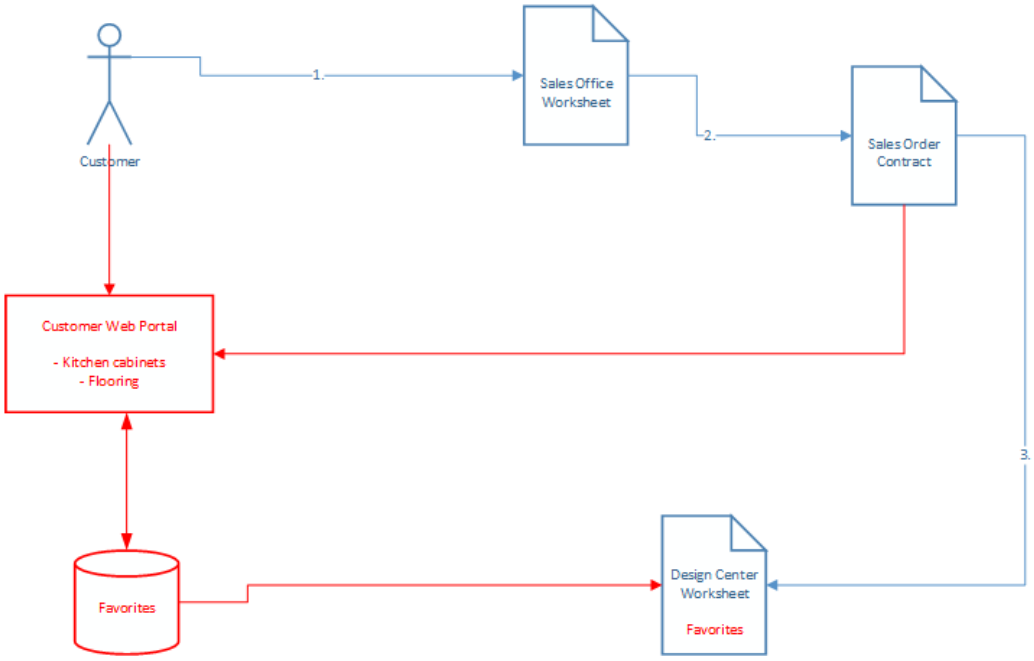


Figure 6.2: Data flow diagram showing the process of a customer buying a home. Parts drawn in blue are already working in the current system. New requirements implemented in this project are drawn in red.

6.2.1 Functional

There are several functional requirements for the web portal. It needs to show design center Option Selections available at the current date for the community and model in the sales order of the customer. It must also show all the available Option Values for attribute and radio Option Selections.

Options can be viewed by category or location. Options can have a picture shown when the option is viewed.

The user can select and save a list of favorite options, styles or colors. These options can be viewed later with sales configurator when visiting the design center.

The web portal needs to be able to be included as a part of the current web site of a customer.

The web service calls need to be accessible by the home builders if they want to make their own version of the web portal.

6.2.2 Usability

Options should be shown in a table layout instead of a flat list so that multiple different options can be compared more easily.

6.2.3 Performance

The time to open the page initially should take less than ten seconds and the time to load the page after each user interaction must be less than one second. The pictures can load individually after that so the page will appear ready even though the pictures are still loading.

6.2.4 Scalability

The web portal should handle loading of 500 Option Selections and 2000 different Option Value, style and color combinations for a single Option Selection. Performance requirements should be satisfied even with the increased data.

6.2.5 Security

The web service should be secure so that other companies cannot access the data of the home builders. It should also be secure enough that the private data of the customers is safe from attackers.

6.2.6 Future Needs

There are some future needs to consider when developing the portal. The smart client application of the sales configurator will be replaced with a website implementation. It would be good if the same calls, that are used with this web portal, could be used with the future version of the sales configurator. That version also needs to show binary and quantity type Option Selections.

Showing statistics about what customers select as favorites would be important for home builders. Though, that is not required for this first phase of implementation.

Chapter 7

Design and Implementation

In this chapter, we describe the design and implementation of the web portal for this project. First, we plan the implementation, compare different technologies and choose which ones to use. After that, we describe more thoroughly how we implemented the web portal.

7.1 Implementation Plan

There are three different parts in the web portal project: server, API and client user interface. The server handles the data and business logic, i.e., receiving and updating the data in the database, representation of that data as objects and different business processes related to that data. That part mostly does not need to be changed for this project. The only change that needs to be made is related to how the data is retrieved for the API. As mentioned in section 6.1.3, the data is right now retrieved in a big chunk. Requirement section 6.2.3 states that the initial load should take less than ten seconds. Right now, it could take more than 30 seconds with large home builders. It would also be more user friendly if the portal remained usable all the time and the loading of the data would not take long. So, the data should be cached or loaded in smaller parts instead of one large chunk.

7.1.1 Application Programming Interface (API)

The API handles the communication between the server and the client. In chapter 3, we described two API technologies: Web Services and Web API. We concluded that Web API is simpler to use than SOAP based Web Services. This is important because a functional requirement in section 6.2.1 states that the API must be accessible for home builders. As home builders do not

necessarily have a web development team, the API should be as simple to use as possible.

A study by Belqasmi et al. [7] compares the performance of SOAP based Web Services and REST style Web API. Web API method has significantly smaller delays and network loads than the Web Service method. The difference is caused by the SOAP message overhead. This matters when considering the performance requirement in section 6.2.3 that requires the delay after each user interaction to be less than a second.

Pautasso et al. [31] compare the architectural aspects of SOAP based Web Services and REST style Web APIs in their paper. They suggest that Web API should be used for ad hoc integration and web services in enterprise application integration projects requiring long lifespan and reliability. However, the flexibility and evolvability of Web API is considered important because of the future development requirement in section 6.2.6 stating that the API calls should support the future implementation of the sales configurator.

Based on these conclusions, we plan to use Web API in this web portal project. Furthermore, we should also consider how to use the REST architectural style and the RMM guidelines. All the REST constraints and RMM levels seem beneficial for this project except for the optional REST constraint code-on-demand.

7.1.2 User Interface (UI)

There are two important factors in creating a user interface for a web portal – client framework and user interface design. The framework was discussed in chapter 2 and user interface design in chapter 4.

7.1.2.1 Framework

The reviewed frameworks are ASP.NET Web Forms, Angular and React. Using ASP.NET Web Forms would be the easiest solution since the rest of the ERP system is already implemented with that. However, a functional requirement in section 6.2.1 states that the portal needs to be included in a customer web site. This would mean that also the customer web site would have to be implemented with ASP.NET Web Forms because the pages are processed on the server side and cannot be included inside a HTML page.

Angular and React are client-side JavaScript libraries which means that applications developed with them are easier to include as a part of a web site. A thesis [30] compares the maintainability and general differences between Angular and React. The conclusion is that there are no significant differences

in maintainability of the frameworks. In addition, Angular is more suitable for large enterprise projects and React for smaller projects. The web portal created for this thesis is a relatively small project and the intent is not to create a whole enterprise web application using the framework of choice.

Furthermore, a study [29] found that React framework can load 2000 items in a second. This is the number of different Option Values that the portal should support by scalability requirement in section 6.2.4, and the page load time in performance requirement section 6.2.3.

In conclusion, we plan to use the React JavaScript library for implementing the user interface.

7.1.2.2 User Interface Design

One requirement in section 6.2.1 is to have the portal included as a part the customer's website. A website design principle discussed in section 4.1 notes to keep the UI consistent. This means that the customer should be able to customize at least the font and colors used on the portal to match their website. This can be achieved by using CSS mentioned in section 2.1. We should define the areas where background colors can be changed and fonts defined. The customer can then edit a separate CSS file where colors and fonts are defined for those areas.

Navigation and layout of the page is something that the customer should not need to worry about. Section 4.2 mentions that the navigation of a website should be either top or left side. Primary navigation on customer websites is usually top side. The web portal can be shown inside the customer website so the navigation can be considered as secondary. Secondary navigation can be on the left side or another row of horizontal menu on top side. It can be difficult to match the look and feel of the primary navigation with another menu row while keeping the site consistent. Therefore, the web portal navigation should be left side.

It is also noted in section 4.2 that left side navigation is better if there are many items and the item names can be long. The left side navigation should contain the categories and option selections. A single category can contain many option selections and the names can be long. The background color of the selected item in navigation should change so the user knows which category and option selection is currently selected.

The option values have an image related to each one of them and that is the most important aspect the user wants to see. Therefore, the values should be shown as an image gallery. The gallery should fill the rest of the page on the right side of the navigation. It should show as many columns and rows as possible while keeping the images large enough. The number of

columns should depend on the size of the browser window.

Another website design principle in section 4.1 states that the navigation should be kept to three clicks. Three click navigation can be achieved if first click is selecting a category, second is selecting an option and the third click is selecting a value as a favorite. The categories should be visible when the page loads for the first time so that showing the categories does not consume a click.

7.1.3 Security

There are two aspects in the security of the web portal to consider. There is the aspect that how the user can be sure that their personal information is safe. The other aspect is how a company can be sure that their data is safe from competitors. These are the two security requirements mentioned in section 6.2.5. This means that the API and the portal need to be secure from unauthorized access. Access Control and methods for implementing it was discussed in chapter 5. In addition, common security risks were discussed in section 5.1.

Authentication for the customer portal will be done as a login with username and password. Remembering yet another password is a burden for the user. Therefore, it would be ideal to use single sign-on for the login that was discussed in section 5.4. Many people have a Facebook or a Google account so either one or both would be a popular choice to use for an OAuth 2 based single sign-on.

After the login, a security token will be used to authenticate user so that the user does not need to enter the username and password multiple times. This token based authentication was reviewed in section 5.3. The token will be tied to the customer. This means that a token authorizes a user to access the data of a single customer.

The login API call will take the username and password as parameter and it will return the security token. Rest of the API calls will require the token to access them. As mentioned in section 5.2, HTTPS should be used to keep the transmitted tokens and passwords safe from attackers. This will keep the personal information of the customer safe if they can store their password safely.

If we make sure that a single token can only access the data related to a single customer, the scope of the data is narrow enough to be no use to the competitors. An API call should only return the necessary subset of the data and only the fields that can be shown to the customer.

This should take care of broken authentication, sensitive data exposure and broken access control from the common security risks. Rest of the secu-

rity risks that need to reviewed for this project are injection, XML external entities, cross-site scripting and insecure deserialization. Security misconfiguration, using components with known vulnerabilities and insufficient logging and monitoring are part of the server maintenance that the administrator should handle.

7.2 Software Implementation

The software implementation for this project can be divided into three parts: server, Web API and user interface. It is clearer to explain the implementation from top to bottom and start from what the user sees and end with what is happening under the hood. First, we will show what was implemented for the user interface of the portal. Second, we will describe how the Web API was implemented. Last, we will explain what changes were required to the source code of the server and the ERP portal.

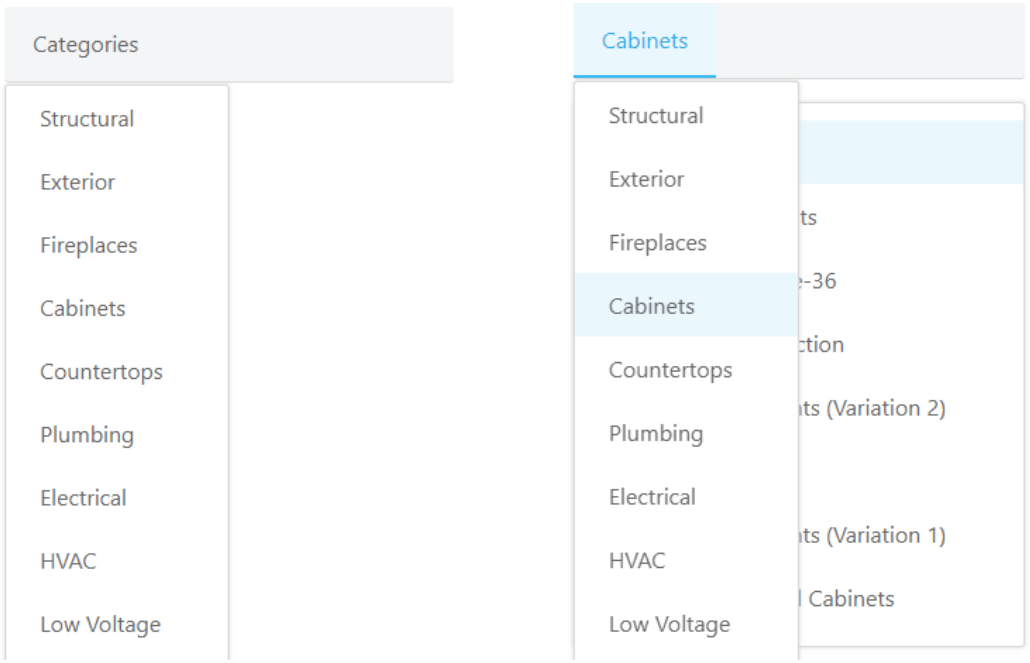
7.2.1 User Interface

When starting the Design Center Portal, the user can choose an existing Wish List or create a new one. The options selected as a favorite are saved to a Wish List Option record tied to the Wish List on which the user is currently working.

As was planned in section 7.1.2.2, the page is divided in left side navigation panel and main content on the right side. The proportion of the left side menu compared to the right side main content is handled by a relative length unit called em. Size of em [6] is relative to the font size where one em is the height of font used in the element. This way the size of an element increases as the user zooms in to increase the font size. If the left panel width was defined with pixels, the width would be too small with larger text or too large with smaller text. If the width was a percentage of the window width, the menu panel width would be too large with larger window size and too narrow with smaller window size.

A third party open source React module is used for the left side menu. On the top of the left side pane is a horizontal menu opening downwards. This menu shows the categories and it is already open when the page loads for the first time. When hovering over a category, the background color in its surrounding area changes. When the user clicks or taps a category, the menu closes and a list of options for that category are shown below the category menu. In addition, the text on the menu button that opens and closes the menu changes to the name of the selected category. If the user hovers over or

clicks the category menu button, it will open again, and the selected category is shown with a different background color.



(a) Initially open menu.

(b) Open menu after selecting Cabinets.

Figure 7.1: Left side menu.

We found out by testing that a good width for the left menu panel is 20 ems to fit most category and option names. If a category name is longer than that, the menu will overflow to the main content. This is fine since the menu is shown only temporarily. The option names will wrap onto the next line because they are shown constantly.

List of options show a horizontal list of unique Option Menus for the Option Selections in a category. This way the user does not need the select a favorite choice multiple times from the same set of styles and color, for example, the flooring choices might be the same for different bedrooms, foyer, flex room and living room. Hovering over an Option Menu or selecting one changes the background color of that option. Selecting an Option Menu shows the available Option Value, style and color choices as a gallery on the right-side panel.

Each choice has an image on the top and a name on the bottom. Name is a combination of the Option Value name, style and color. If a style and color choice does not have an image available, a missing image picture is shown

instead of the choice image. Height of the name is set to be 2 em so it shows two lines of text for the name.

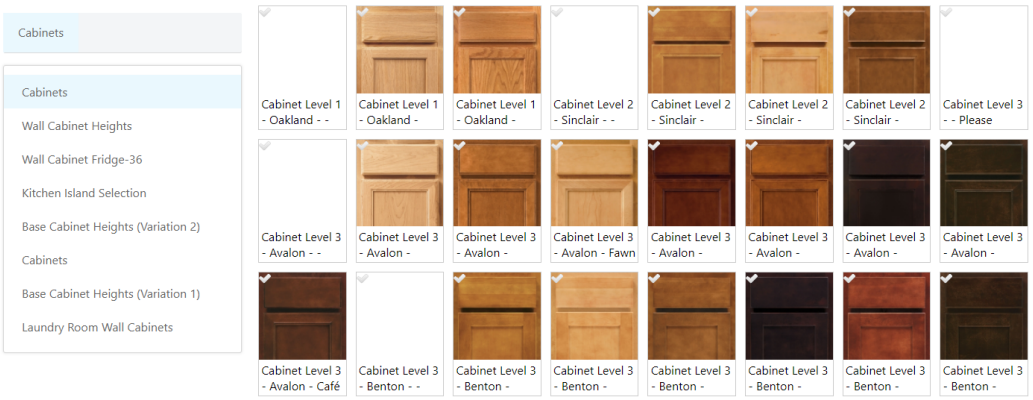


Figure 7.2: Image showing Cabinet choices.

Because the name does not necessarily fit into two lines, the user can click the choice to see the name fully, a full-size image and description about the choice. This detailed view fills the whole main content panel. The user can click it again to return to the gallery view.

The user interface is designed to work with desktop computers and mobile devices as well as different sizes of viewport, i.e., the screen or the browser window. The choice gallery shows different number of columns of images based on the viewport width. A maximum of eight columns are shown when the page width is larger than 1200 pixels. Four columns are shown when the width is larger than 800 pixels and two columns when the width is larger than 600 pixels. Just one column is shown when the width is smaller than that. This way the images stay large enough to see them while still having as many images as possible visible at the same time.

Top left corner of the choice has an image that can be clicked to select the choice as a favorite. Selecting a favorite choice marks the selections completed for the Option Menu in the list. Completed Option Menu has an icon on the left side of the name. Once favorites are selected for each Option Menu in a category, the category will be marked as completed with an icon in the category menu. If some but not all Option Menus in a category are completed, the category has an icon marking it partially completed.

7.2.2 Web API

The login Web API call for the Customer Portal was implemented related to another master's thesis for our company [32]. The call takes username and password for a customer as parameters and returns a security token. We can use that token with the Web API calls of this project to authenticate the user. The Customer is tied to that token so for each call we can get the Sales Order for the Customer. As is described in section 6.1.4, The Sales Order contains information which Community, Model and Sales Office options the customer has selected. This information is needed to know which categories and options are available for the user.

Four Web API calls are needed to get the information about the options to show. First one is for fetching the categories. Second call gets the Option Menus for a category. Third call retrieves Option Value, Style and Color choices in the Option Menu. Fourth one gets the image for the choice. This division of Web API calls was decided because this is also the information that needs to be retrieved after each user interaction. In other words, after a user selects a category or an option, new information needs to be loaded based on that selection. The image fetching call is separated from the Option Value call because loading of the images takes more time. The page appears to load faster if the choice data is shown first and the images load afterwards dynamically one by one.

The Category Web API call returns available categories for the customer. The URI is `/api/categories`, and it returns an object with string type fields OptionCategoryID and Name. No additional parameters are required since the customer and related information can be fetched from the header security token.

The Option Menu Web API call takes the selected OptionCategoryID as a parameter and returns available Option Menus in that category. The URI of the call is `/api/categories/<OptionCategoryID>/options`, for example, `/api/categories/CAB/options` returns cabinet options. The Option Category ID is unique in the database, so it can be used as a parameter in the call. The returned object has fields OptionID and Name, both of which are string type.

The Option Value Web API call returns available choices for the selected OptionID parameter. The URI is `/api/optionmenus/<OptionID>/choices`, for example, `/api/optionmenus/CAB-STD/choices`. The Option Menu ID is unique for a Model. The calls handle Option Menus only from a single Model since the calls are tied to a Customer who has already selected the Model for the Home. Therefore, the Option Menu ID can be used as the parameter for this call. The Web API call accepts string as the Option ID parameter, so it

is easier to change later to something else if needed. The return object has string type fields OptionID, OptionValueID, Name, Style and Color.

The image Web API call takes the OptionID, OptionValueID, Style and Color as parameters and returns a jpeg image for that choice. If the image does not exist, a HTTP status code 404 is returned instead. The call URI is `/api/optionmenus/<OptionID>/choices/<OptionValueID>~<Style>~<Color>`, for example, `/api/optionmenus/CAB-STD/choices/cab-3~Avalon~Saddle` for level three saddle color Avalon style kitchen cabinet. The Style or Color might be empty for some options. Then the call could be `/api/optionmenus/CAB-STD/choices/cab-1~~`.

The operations to handle Wish Lists need seven new Web API calls. Four are related to the Wish Lists. One is needed for fetching the existing Wish Lists, the second one for creating a new Wish List, the third one for changing its name and the fourth one for removing it. Three calls are needed for the Wish List Options. One call is required for getting existing Wish List Options, another one for updating a favorite choice to a Wish List Option and the last one for removing a favorite choice.

As was discussed in section 3.4, multiple HTTP verbs should be used to for different operations to the same resource instead of using multiple URIs. Furthermore, HTTP Semantics and Content RFC [17] instructs us on the use of HTTP verbs PUT, POST and DELETE. PUT should be used create or replace a resource at a URI. If the client does not know what is the URI of the resource to create, POST should be used instead to create that resource. Then, the server can decide what is the URI of the new resource and the response for the POST request will return that URI. That specific URI can then be used to update a field in that resource with PUT or delete the resource with DELETE. PUT and POST should have the fields to create or update in the body of the request. DELETE requests should not have a body.

The URI to get existing Wish Lists is `/api/wishlists`. The call returns objects with fields WishListID and Name. To create a new Wish List a POST request is send to the same URI `/api/wishlists`. Name of the new Wish List is sent in the body of the request. Returned object contains the new WishListID and the Name. The URI of the new Wish List is returned in the Location header as `/api/wishlists/<WishListID>`. That URI can be used in a PUT request to update the Name or DELETE request to remove the Wish List. Body of the PUT request should contain the new Name.

Existing favorite choices for a Wish List can be fetched with a call to URI `/api/wishlists/<WishListID>/options`. It returns a list objects with fields OptionID, OptionValueID, Style and Color. A choice can be marked as favorite with a PUT request to URI `/api/wishlists/<WishListID>/options/`

`<OptionID>`. The body of the request should contain OptionValueID, Style and Color for that choice in the Option Menu identified by the OptionID. The same URI can be used to deselect a favorite choice with a DELETE request.

All the objects returned by the Web API calls are in JSON format by default. This way the objects are easy to handle in JavaScript. User of the API can alter the request header to get the return value in XML format.

7.2.3 Server

Initial plan was to divide the fetching of the data into smaller chunks to save time on the initial load. This turned out to be much more difficult tasks than anticipated. Due to time constraints, we decided to use a cache instead.

New business objects to create for this project are Wish List and Wish List Option. Wish Lists are tied to a customer and a sales order, and it has a name that can be specified by the user. Wish List Option record is tied to one Wish List and it defines the favorite Option Value, style and color for an Option Menu. Wish List can have multiple Wish List Options.

The common security risks that need to be taken care of for this project are injection, XML external entities, cross-site scripting and insecure deserialization. Injection in the context of this project is specifically SQL injection. This issue is solved in the SQL query adapter that is used by the system since it uses parameterized queries. XML external entities is not an issue because the portal uses Web API instead of web services and SOAP. Cross-site scripting attacks are handled by the React JS library. Insecure deserialization is not an issue because the state of the portal is not serialized. In addition, the parameters for the Web API are simple data types so they are not vulnerable.

Chapter 8

Results and Discussion

In this chapter we evaluate how well did the implementation of this project succeed, discuss about it and how could it be improved in the future. First, we evaluate the implementation and discuss about it. After that, we talk about the future work related to this project.

8.1 Evaluation

This project is evaluated based on the requirements in section 6.2 and based on tests about the speed of the Web API implementation. First, we compare the implementation to the requirements. Second, we discuss about the tests that were done.

8.1.1 Requirements

Almost all of the functional requirements in section 6.2.1 are met. The portal shows available design center options and their choices. Options can be only viewed by category. Grouping by location was not implemented since it was not deemed necessary at this point. Favorite choices are saved in a Wish List and each choice can have picture. The React JavaScript UI library allows the web portal to be included in a web site of a home builder as was described in section 7.1.2.1. To allow further customization, CSS can be used to change the font, colors and certain images used. This was described in section 7.1.2.2. As was planned in section 7.1.1, the Web API is easy to use by home builders if they want create their own portals.

The usability requirement in section 6.2.2 stating that the options should be shown in a table layout is fulfilled since the choices are shown as a gallery.

The scalability requirement in section 6.2.4 states that the portal should

handle loading of 500 Option Selections and 2000 choices for a single Option Value. During the development the Option Selections changed to Option Menus to minimize duplicate choices. This requirement is tested in the next section 8.1.2.

The security requirement in section 6.2.5 requires the personal information of the customers and the data of the home builders are safe. The portal was planned and implemented keeping the security in mind. This was discussed in planning section 7.1.3 and implementation section 7.2.3.

8.1.2 Tests

The performance requirement in section 6.2.3 requires that the initial page load is less than ten seconds and the load after each user interaction is less than a second. The user interface handles the loading quickly and the bottleneck in this case is the Web API. It can be tested with a program called Postman [34]. It can create HTTP request with different verbs, headers and body. It shows what the result is and how long did it take to process. There are also tests that can be created with it.

We tested loading of categories, options and choices with Web API calls `/api/categories`, `/api/categories/<OptionCategoryID>/options` and `/api/optionmenus/<OptionID>/choices`. The tests were carried out with data sets from three different home builders. Let us call them home builders A, B and C. Each call was run four times for each data set.

All the data sets are backups from environments that are used in production. They contain actual data that home builders use to sell homes and options to customers. Home builder A has just recently started using the system. Builders B and C have used the system for a few years but the data set of home builder C is not built optimally to support large amount of data.

The data set sizes and average load times of the tests are shown in the results that are shown in table 8.1. Data sizes of categories and options is the total number of those objects in the data set available for a model. The data sizes of choices are the maximum number of choices found for an option menu in the data set. The tests were run for that option menu and the category in which the option menu is. The model was chosen so that it has the option with the maximum number of choices and the maximum number of other options available.

The data set sizes in table 8.1 show that they are not large enough to test what was required in scalability requirement in section 6.2.4. Home builder C has the largest number of options and choices that are 715 and 931. The requirement is to handle 500 options and 2000 choices. The number of options is enough to test the scalability but the number of choices is less than half of

Table 8.1: Loading time tests with data sets from home builders A, B and C.

	A data size	A load time (ms)	B data size	B load time (ms)	C data size	C load time (ms)
Categories	16	539	19	2126	18	19339
Options	320	2127	404	3583	715	33030
Choices	134	2131	504	3602	931	32888

what was required. This is fine for now since there are not any bigger data set being used but it might become an issue in the future. So, it should be tested at some point.

Bigger issue is that none of the load times meet the page load performance requirement in section 6.2.3. Even the smallest data set of home builder A has page loads of more than two seconds for loading options or choices. The initial page load of less than ten seconds is only an issue in the data set C. Home builders A and B have the initial category page load time clearly less than ten seconds. The category load time is less than the other load times for all the data sets because it takes less time to process the option availabilities on the server side. The option availabilities based on the selected options needs to be processed for the options and choices but not for the categories.

The figure 8.1 shows how the page load time increases exponentially with the data set size. This means that the current cache utility is not a viable solution for this portal. How to improve the load time was not researched for this thesis as it was not foreseen to become an issue for the portal. This is discussed further in the future work section 8.2.

8.2 Future Work

We did not implement one functional requirement from section 6.2.1, namely, grouping the options by location. This can be considered something to do in the future. The sales configurator currently support grouping by location, so this needs be implemented when the sales configurator will be replaced by a web site as discussed in future requirements in section 6.2.6. That will also need to show binary and quantity type Option Selections. Changing to current Web API to support that should be easy. More changes are needed to the UI but we selected React so that adding new parts to the UI should be easy as well.

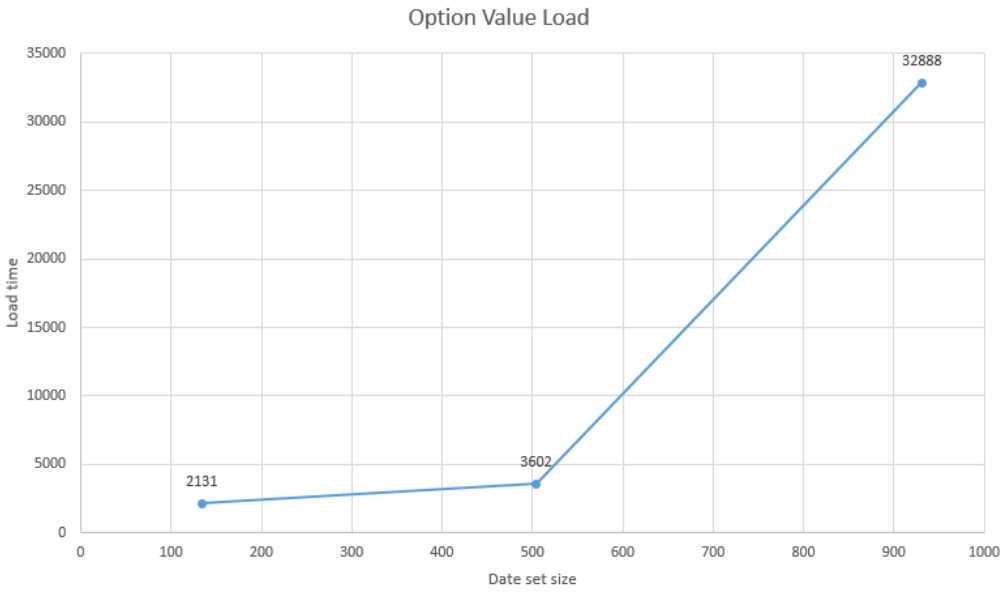


Figure 8.1: A line chart showing load time of choices in relation to the number of choices.

As is discussed in section 8.1.2, the page load times are not fast enough. So, either the cache needs to be improved for the portal or the data needs to be loaded in smaller sets. Improving the cache might be good enough since it stores a lot of unnecessary data not needed for this portal. In addition, the data set used to test this was not big enough so additional choices should be created for the tests. This should be done when optimizing the cache to make sure it works with bigger data sets.

Other future requirements were discussed in section 6.2.6. The current Web API can be easily changed to support the future version of the sales configurator and show all types of options. In addition, showing statistics about favorite options should be easy since they are saved in the separate Wish List Options table in the database. A report can be created from the data in the table.

Chapter 9

Conclusions

The goal of this thesis is to implement a customer portal for home buyers to select options for their homes. The first part of thesis consists of a literature review about the technologies needed to implement the portal. The second part entails implementation plan, implementation and evaluation.

The reviewed technologies are related to web application framework, application programming interface, user interface design and security. The reviewed frameworks are ASP.NET, Angular and React. In ASP.NET, the web page is programmed in two parts. One file contains description about the layout and the other contains the business logic written in C#. Angular and React are client-side JavaScript libraries. They both offer methods for updating the UI based on changes in the data.

The reviewed application programming interface technologies are SOAP, Web Services, REST and Web API. Web Services are application functionality that can be accessed over the network. Web Services architecture can be described by layers called discovery, description, packaging, transport and network. SOAP is a data format used for the packaging layer. Web API is an interface for accessing data using standard HTTP methods and headers. REST is a commonly used architectural style for designing Web API.

User interface design tips and guidelines are given about website design, navigation and page design. Security chapter reviews common security risks, access control, token-based authentication and single sign-on.

In the implementation plan, we selected React from the frameworks for the implementation because of its performance and suitability for small projects. In addition, Web API is used as the application programming interface due to its evolvability, flexibility, performance and ease of use.

The most useful user interface design guidelines for this project were that the navigation should be kept to three clicks and the navigation should be left or top side. Furthermore, a temporarily shown modal menu is efficient if

there is a lot of information to show.

The security risks that are most related to this project are access control vulnerabilities, SQL injection and cross-site scripting. Access control risks can be reduced with single sign-on. SQL injection attacks can be prevented by using parameterized queries. Cross-site scripting attacks can be countered by a framework that handles them.

The issues encountered during the project were related to the performance of the API. This is caused by the old implementation of how the underlying data is fetched. However, optimizing it was not part of the project, and it was not researched. Though, this is something that needs to be solved in the future as customers start to use the portal.

Ultimately, the project succeeded well – the framework, the API, the user interface design and the security risks were researched properly, and they were useful in the planning and the implementation of the project. Furthermore, React JavaScript library and Web API can be recommended for anyone implementing a web portal. In addition, the aforementioned user interface design and security guidelines can be useful for web application development.

Bibliography

[1] Angular. `https://angular.io/`. Accessed: 2016-12-19.

[2] Babel. `https://babeljs.io/`. Accessed: 2016-10-27.

[3] jQuery. `https://jquery.com/`. Accessed: 2016-12-19.

[4] React. `https://facebook.github.io/react/`. Accessed: 2016-12-19.

[5] Shibboleth identity provider. `https://www.shibboleth.net/products/identity-provider/`. Accessed: 2017-08-22.

[6] Atkins, T., and Etemad, E. CSS Values and Units Module Level 3. Candidate Recommendation, W3C, Sept. 2016. https://www.w3.org/TR/2016/CR-css-values-3-20160929/.

[7] Belqasmi, F., Singh, J., Melhem, S. Y. B., and Glitho, R. H. Soap-based vs. restful web services: A case study for multimedia conferencing. *IEEE Internet Computing 16*, 4 (July 2012), 54–63.

[8] Block, G., Cibraro, P., Felix, P., Dierking, H., and Miller, D. *Designing Evolvable Web APIs with ASP.NET*. O'Reilly Media, Inc., 2014.

[9] Bonelo, E. B. OpenID Connect Client Registration API for Federated Cloud Platforms. Master's thesis, Aalto University, School of Science, Department of Computer Science, 2017.

[10] Butow, E. *User Interface Design for Mere Mortals*. Addison-Wesley, 2007.

[11] Cardis, P. Why the internet drives housing. *Builder 35*, 11 (Nov. 2012), 40.

[12] Chen, J., Cui, X., Zhao, Z., Liang, J., and Guo, S. Toward discovering and exploiting private server-side web apis. *IEEE International Conference on Web Services* (2016).

[13] City of Helsinki. Helsinki developers' portal – APIs. `https://dev.hel.fi/apis/`. Accessed: 2017-03-25.

[14] Clarke, J. *SQL Injection Attacks and Defense*. Elsevier, 2012.

[15] Cooper, A., Reimann, R., and Cronin, D. *About Face: The Essentials of Interaction Design*. John Wiley & Sons, Inc., 2014.

[16] Fett, D., Küsters, R., and Schmitz, G. The web SSO standard openid connect: In-depth formal security analysis and security guidelines. *CoRR abs/1704.08539* (2017).

[17] Fielding, R., and Reschke, J. Hypertext transfer protocol (HTTP/1.1): Semantics and content. RFC 7231, Internet Engineering Task Force (IETF), June 2014. `https://tools.ietf.org/html/rfc7231`.

[18] Fielding, R. T. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[19] Fielding, R. T., and Taylor, R. N. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT) 2*, 2 (2002), 115–150.

[20] Flanagan, D. *JavaScript: The Definitive Guide*. O'Reilly Media, Inc., 2011.

[21] Hatvala, A. Open Innovation Opportunities and Business Benefits of Web APIs: A Case Study of Finnish API Providers. Master's thesis, Aalto University, School of Business, Department of Information and Service Economy, 2016.

[22] Huluka, D., and Popov, O. Root cause analysis of session management and broken authentication vulnerabilities. In *World Congress on Internet Security (WorldCIS-2012)* (June 2012), pp. 82–86.

[23] Kozlowski, P., and Darwin, P. B. *Mastering Web Application Development with AngularJS*. Packt Publishing, 2013.

[24] Larsen, R. *Beginning HTML and CSS*. Wiley, 2013.

[25] Le Hors, A., Le Hégaret, P., Wood, L., Nicol, G., Robie, J., Champion, M., and Byrne, S. Document Object Model (DOM) Level 3 Core Specification. `https://www.w3.org/TR/DOM-Level-3-Core/`, Apr. 2004. Accessed: 2016-11-17.

[26] Lee, R. B. *Security Basics for Computer Architects*. Morgan & Claypool, 2013.

[27] Li, S. React: Create maintainable, high-performance UI components. `https://www.ibm.com/developerworks/library/wa-react-intro/index.html`, 2015. Accessed: 2017-11-01.

[28] Lynch, P. J., and Horton, S. *Web Style Guide: Basic Design Principles for Creating Web Sites*. Yale University Press, 2008.

[29] Molin, E. Comparison of Single-Page Application Frameworks: A method of how to compare Single-Page Application frameworks written in JavaScript, 2016.

[30] Mousavi, S. A. Maintainability Evaluation of Single Page Application Frameworks: Angular2 vs. React, 2016.

[31] Pautasso, C., Zimmermann, O., and Leymann, F. Restful web services vs. "big" web services: Making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web* (New York, NY, USA, 2008), WWW '08, ACM, pp. 805–814.

[32] Phan, M. Web Application Programming Interface Design for a Customer Portal. Master's thesis, Aalto University, School of Science, Degree Programme in Computer Science and Engineering, 2015.

[33] PlanMill. PlanMill API documentation. `https://developers.planmill.com/api/`. Accessed: 2017-03-25.

[34] Postdot Technologies, Inc. Postman – Supercharge your API workflow. `https://www.getpostman.com/`. Accessed: 2017-10-15.

[35] Schell Brothers, LLC. Schell Brothers – Design Your Home. `http://schellbrothers.com/design/`, 2016. Accessed: 2016-11-22.

[36] Snell, J., Tidwell, D., and Kulchenko, P. *Programming Web services with SOAP*. O'Reilly Media, Inc., 2001.

[37] Spaanjaars, I. *Beginning ASP.NET 4: in C# and VB*. Wiley, 2010.

[38] Stefanov, S. *React: Up & Running: Building Web Applications.* O'Reilly Media, 2016.

[39] Sulatycki, R., and Fernandez, E. B. Two threat patterns that exploit "security misconfiguration" and "sensitive data exposure" vulnerabilities. In *Proceedings of the 20th European Conference on Pattern Languages of Programs* (New York, NY, USA, 2015), EuroPLoP '15, ACM, pp. 46:1–46:11.

[40] Suoranta, S. Enhanced security for mobile user authentication and single sign-on, 2016.

[41] Tarasiewicz, P., and Böhm, R. *AngularJS.* Brainy Software, 2014.

[42] Tasse, D., Ankolekar, A., and Hailpern, J. Getting users' attention in web apps in likable, minimally annoying ways. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2016), CHI '16, ACM, pp. 3324–3334.

[43] Torbica, Ž. M., and Stroh, R. C. Customer satisfaction in home building. *Journal of Construction Engineering and Management 127*, 1 (Feb. 2001), 82–86.

[44] van der Stock, A., Glas, B., Smithline, N., and Gigler, T. *OWASP Top 10 2017: The Ten Most Critical Web Application Security Threats Release Candidate 2.* Open Web Application Security Project, 2017.

[45] Vohs, K. D., Baumeister, R. F., Schmeichel, B. J., Twenge, J. M., Nelson, N. M., and Tice, D. M. Making choices impairs subsequent self-control: A limited-resource account of decision making, self-regulation, and active initiative. *Journal of Personality and Social Psychology Vol. 94*, No. 5 (May 2008), 883–898.